Week 4 – Flask Microframework

Monday

- Setup Flask Project
- Routing
- Jinja2 Templates & How They Work
- **Exercise**: Commit Flask project to GitHub

Tuesday

- Flask Bootstrap
- Organizing file structure
- Mixins
- Decorators
- **Workshop**: Control flow with Jinja2

Wednesday

- Flask WTF (forms)
- MVC and web app design
- Flask Database Setup with SQLite
- **Workshop**:
    - Flask app using web form for login page
    - Web application refactor

Thursday

- Registration & Login
    - User Authentication
    - Password Hashing
- Database Migration
- User Query
- **Exercise**:
- **Workshop**: Login/Registration Page Working with SQLite Database

Friday

- **Interviews:** 10 minutes per person
- **Project:** Convert NZA Law website to a Flask Application
- **Weekend Challenge**:
    - Finish Converting NZA Law website to a Flask Application (commit)

Flask Basic

What is Flask

Flask is a **microframework** for Python based on Werkzeug, Jinja2.

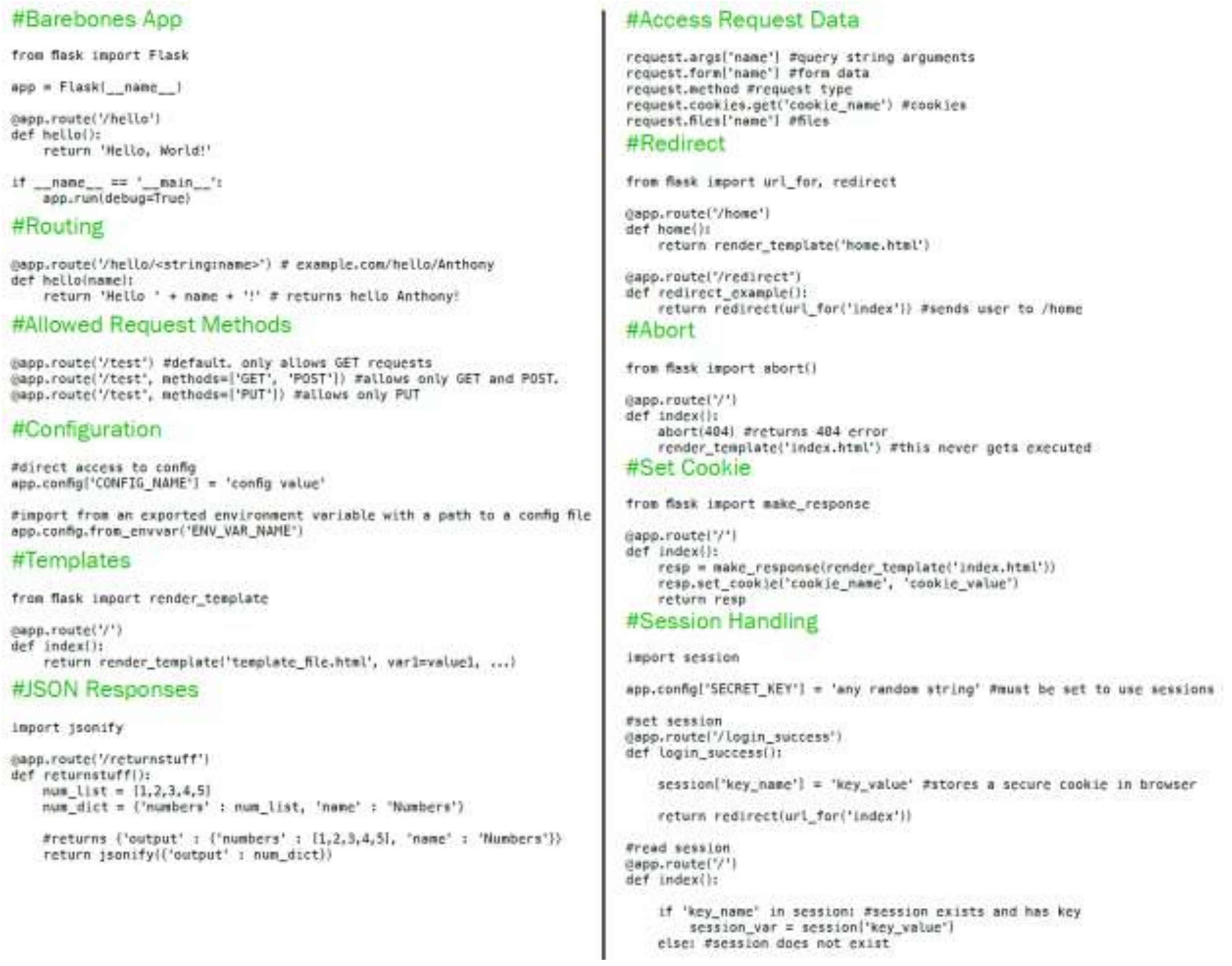
- **Werkzeug** is a utility tools for the Web Server Gateway Interface (WSGI) is a specification for simple and universal interface between web servers and web applications or frameworks for the Python programming language.
- **Jinja2** is a modern and designer-friendly templating language for Python

Routing

In Flask The route() decorator is used to bind a function to a URL.

```
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello, World'
```

Flask Cheat Sheet



#Barebones App

```
from flask import Flask

app = Flask(__name__)

@app.route('/hello')
def hello():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(debug=True)
```

#Routing

```
@app.route('/hello/<string:name>') # example.com/hello/Anthony
def hello(name):
    return 'Hello ' + name + '!' # returns hello Anthony!
```

#Allowed Request Methods

```
@app.route('/test') #default. only allows GET requests
@app.route('/test', methods=['GET', 'POST']) #allows only GET and POST.
@app.route('/test', methods=['PUT']) #allows only PUT
```

#Configuration

```
#direct access to config
app.config['CONFIG_NAME'] = 'config value'

#import from an exported environment variable with a path to a config file
app.config.from_envvar('ENV_VAR_NAME')
```

#Templates

```
from flask import render_template

@app.route('/')
def index():
    return render_template('template_file.html', var1=value1, ...)
```

#JSON Responses

```
import jsonify

@app.route('/returnstuff')
def returnstuff():
    num_list = [1,2,3,4,5]
    num_dict = {'numbers' : num_list, 'name' : 'Numbers'}

    #returns {'output' : {'numbers' : [1,2,3,4,5], 'name' : 'Numbers'}}
    return jsonify({'output' : num_dict})
```

#Access Request Data

```
request.args['name'] #query string arguments
request.form['name'] #form data
request.method #request type
request.cookies.get('cookie_name') #cookies
request.files['name'] #files
```

#Redirect

```
from flask import url_for, redirect

@app.route('/home')
def home():
    return render_template('home.html')

@app.route('/redirect')
def redirect_example():
    return redirect(url_for('index')) #sends user to /home
```

#Abort

```
from flask import abort()

@app.route('/')
def index():
    abort(404) #returns 404 error
    render_template('index.html') #this never gets executed
```

#Set Cookie

```
from flask import make_response

@app.route('/')
def index():
    resp = make_response(render_template('index.html'))
    resp.set_cookie('cookie_name', 'cookie_value')
    return resp
```

#Session Handling

```
import session

app.config['SECRET_KEY'] = 'any random string' #must be set to use sessions

#set session
@app.route('/login_success')
def login_success():

    session['key_name'] = 'key_value' #stores a secure cookie in browser

    return redirect(url_for('index'))

#read session
@app.route('/')
def index():

    if 'key_name' in session: #session exists and has key
        session_var = session['key_value']
    else: #session does not exist
```

Mixins in Python

Python supports a simple type of **multiple inheritance** which allows the creation of Mixins. Mixins are a sort of class that is used to "mix in" extra properties and methods into a class. This allows you to create classes in a compositional style.
Please Note: The correct way to use Mixins is like in the reverse order because the priority of how methods are resolved is from left to right.

```
class Mixin1(object):
    def test(self):
        print "Mixin1"

class Mixin2(object):
    def test(self):
        print "Mixin2"

class MyClass(Mixin2, Mixin1, BaseClass):
    pass
```

Decorator in Python

Decorators are **functions** which modify the functionality of another **function**. They help to make our code shorter and more **Pythonic**.

Example

```
# define the first decorator
def a_new_decorator(a_func):
    def wrapTheFunction():
        print("I am doing some boring work before executing a_func()")
        a_func()
        print("I am doing some boring work after executing a_func()")
    return wrapTheFunction
def a_function():
            print("I am the function which needs some decoration")

@a_new_decorator
def a_function_2():
    print("I am the function which needs some decoration")
```

Call it

```
a_function = a_new_decorator(a_function)

a_function()
```

Same as above

```
a_function_2()
```

Setup Flask Project

Using Atom

- Create a new project using Atom.
- Make sure the project Anaconda environment setup correct
- Check the project setting, make sure the project python interpreter is setup as well.
- Create the basic Flask application

Git

Git ignores files.

Ignore some files, which are not necessary to put in repo.

GitHub and Remote

Create a GitHub project

Adding remote repo, usually we call it Origin, but we can give any name.

Using Branch

Branch the work and git keep all branch information In the same project folder

Run Flask application

1. Using flask commands

```
FLASK_APP=hello.py
```

or for windows

```
set FLASK_APP=hello.py
```

2. Using directly call python file and in code execute app.run()

Flask Basic

What is Flask

Flask is a **microframework** for Python based on Werkzeug, Jinja2.

- **Werkzeug** is a utility tools for the Web Server Gateway Interface (WSGI) is a specification for simple and universal interface between web servers and web applications or frameworks for the Python programming language.
- **Jinja2** is a modern and designer-friendly templating language for Python

Micro framework and Extensions

- Micro framework does less for you and leaves more freedom on how you write and organize your code
- Extensions are a very important part of the Flask ecosystem. They provide solutions to common problems

There are a lots of flask extensions we can use, for example

- flask-wtf
- flask-sqlalchemy
- flask-migrate

- flask-login

Install a package

$ pip install <package-name>

Very Basic Flask App

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
   return "Hello World!"
```

Routing

Decorator In Flask The @app.route() **decorator** is used to bind a function to a URL.

```
@app.route('/')
@app.route('/hello')
```

Http methods

```
@app.route('/login', methods=['GET', 'POST'])
```

Redirect and Generating Links

```
from flask import redirect, flash, url_for
...
redirect(url_for('login'))
```

Jinja Templates

In Flask, templates are written as **separate** files, stored in a **templates folder** that is **inside the application package**.

Render template

The operation that converts a template into a complete **HTML** page is called rendering. To render the template I had to import a function that comes with the Flask framework called `render_template()`. This function takes a template filename and a variable list of template arguments and returns the same template, but with all the placeholders in it replaced with actual values.

Jinja2

Jinja2 is a modern and designer-friendly templating language for Python.

- The render_template() function invokes the Jinja2 template engine that comes bundled with the Flask framework.
- Jinja2 substitutes {{ ... }} blocks with the corresponding values, given by the arguments provided in the render_template() call.

Control Flow

Repeating

```
{% for post in posts %}
        <div>
          <H1>{{ post.title }}</H1>
          {{ post.body }}
        </div>
        {% endfor %}
```

Condition

```
{% if title %}
        <title>{{ title }} - !</title>
        {% else %}
        <title>Welcome!</title>
        {% endif %}
```

Template Inheritance

- Create a base.html that contains the that are common layouts such as navigation bar, header and footer to all other templates.
- In the Base template set placeholder

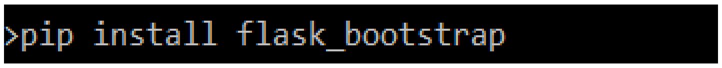
```
{% block content %}{% endblock %}
```

- Other templates extend base.html

```
{% extends "base.html" %}
```

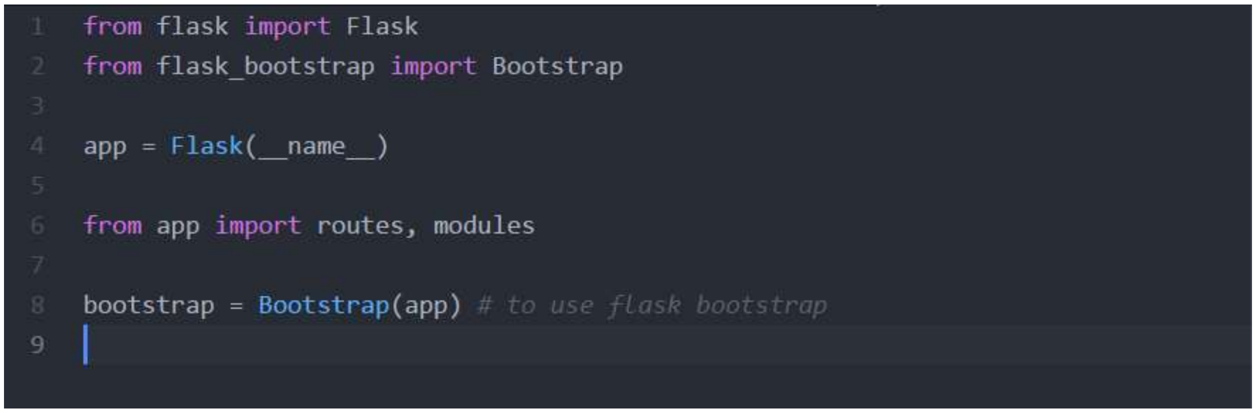
Flask Bootstrap

Install flask_bootstrap



```
>pip install flask_bootstrap
```

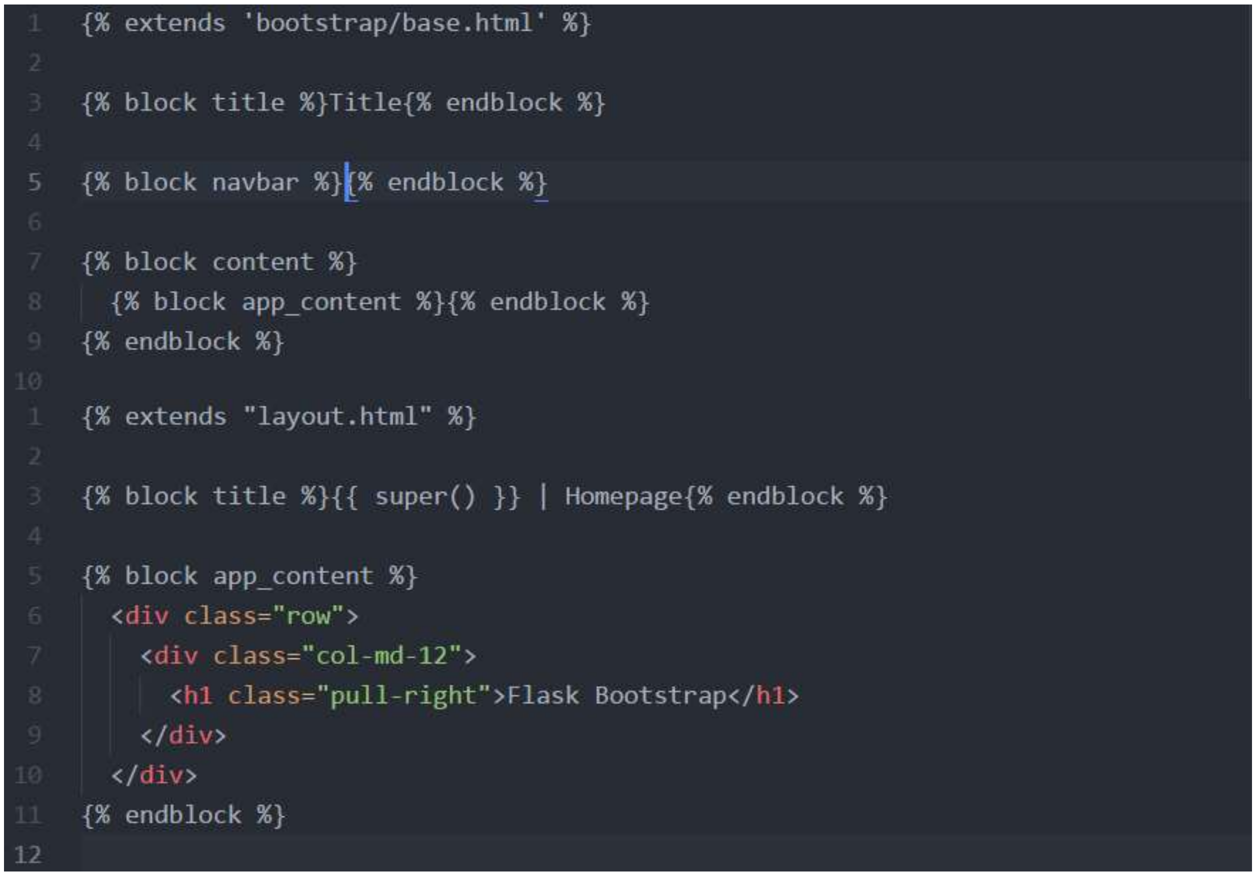
Importing to App



```
from flask import Flask
from flask_bootstrap import Bootstrap

app = Flask(__name__)

from app import routes, modules

bootstrap = Bootstrap(app) # to use flask bootstrap

```

Using flask_bootstrap

- Flask-Bootstrap allows you to incorporate Bootstrap, without the hassle of importing it everywhere
- Extending from the layout allows for ease of use
- One time editing
- Basic documentation:
  - https://pythonhosted.org/Flask-Bootstrap/basic-usage.html
- Key block statements:
  - Block Title
  - Block Navbar
  - Block Content
  - Block App_Content



```
{% extends 'bootstrap/base.html' %}

{% block title %}Title{% endblock %}

{% block navbar %}{% endblock %}

{% block content %}
  {% block app_content %}{% endblock %}
{% endblock %}

{% extends "layout.html" %}

{% block title %}{{ super() }} | Homepage{% endblock %}

{% block app_content %}
  <div class="row">
    <div class="col-md-12">
      <h1 class="pull-right">Flask Bootstrap</h1>
    </div>
  </div>
{% endblock %}

```

Flask bootstrap incorporates the same principles as Bootstrap; however, it allows you to use the "extends" functionality to keep your project clean and easily editable.

Flask WTF

Install the extension

```
install flask-wtf
```

app.config and SECRET_KEY

```
app.config['SECRET_KEY'] = 'you-never-know'
```

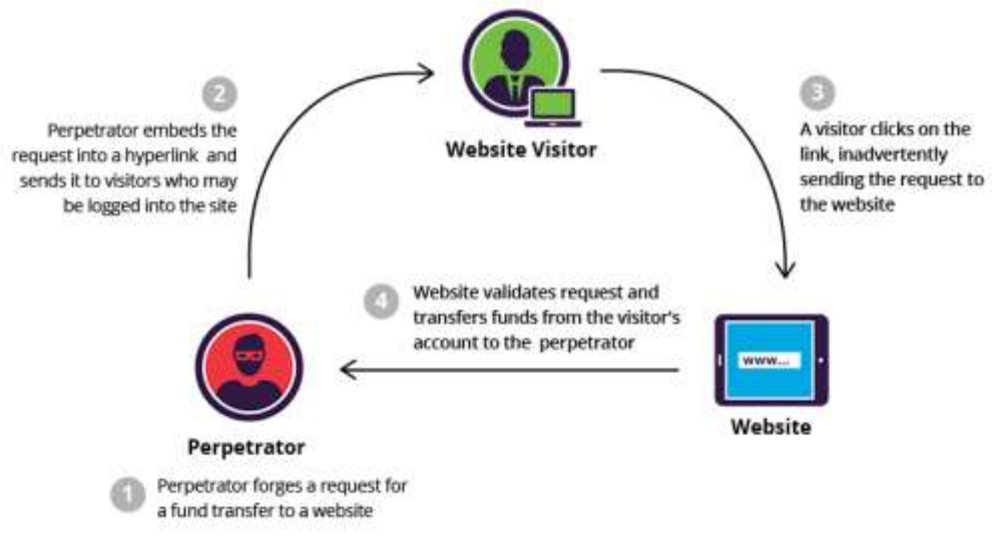
- app.config is **like** a dictionary. We can set and get the value by key.
- **SECRET_KEY** configuration variable is an important part in Flask applications. Flask and some of its extensions use the value of the secret key as a cryptographic key, useful to generate signatures or tokens. The Flask-WTF extension uses it to protect web forms against a nasty attack called Cross-Site Request Forgery or CSRF.

Using flask-wtf

- Create a class named LoginForm extend from (FlaskForm)
- In route create a **form object** and pass to it the template for rendering.
- Using **form object** in the Jinja2 template

Security and Hidden tag

CSRF or XSRF (Cross-site request forgery), also known as one-click attack or session riding (pronounced sea-surf). It is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts.


Website Visitor
2 Perpetrator embeds the request into a hyperlink and sends it to visitors who may be logged into the site
3 A visitor clicks on the link, inadvertently sending the request to the website
4 Website validates request and transfers funds from the visitor's account to the perpetrator
www...
Website
Perpetrator
1 Perpetrator forges a request for a fund transfer to a website

form.hidden_tag() generates a hidden field that includes a token used to protect the form against CSRF attacks according to **SECRET_KEY**.

Receiving Form Data

Get or Post?

- Rendering the page is Get
- Submit form is post

Using `form.validate_on_submit()`

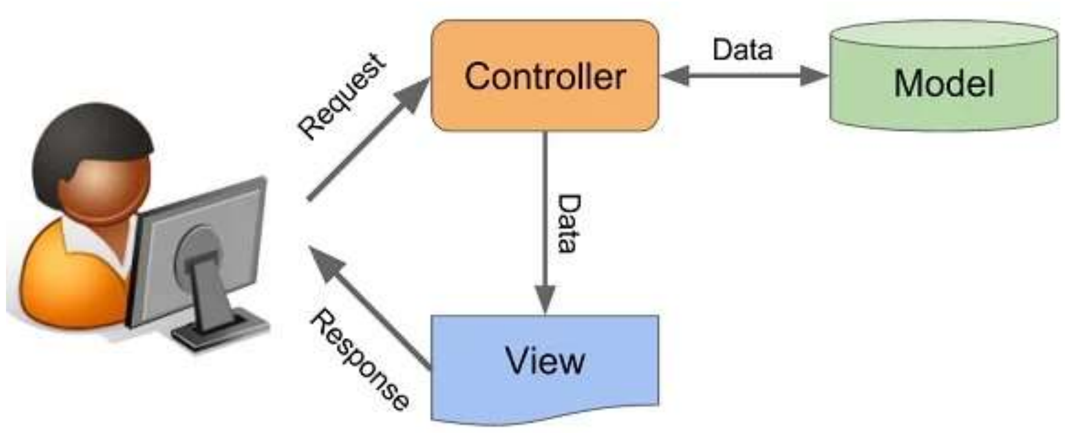
- Checking
- Distinguish http methods

MVC and web app design

MVC

The Model-View-Controller (MVC) is not a technology, but a concept in software design/engineering. The MVC consists of three components, the Model, the View and the Controller

From https://helloacm.com/model-view-controller-explained-in-c/



THE MODEL

The Model is directly responsive for handling data

VIEW

The View component knows how to present the Data to the users.

CONTROLLER

The Controller can ask the Model to update its data. Also, the Controller can ask the View to change its presentation. Basically, it is a component that takes input from the user and sends commands to the View or Model.

Refactoring project

- Arrange code and put file to different modules.
- Follow common designs.

Flask Database

Database

- Storing data.
- Special designed application.

SQLAlchemy

Flask-SQLAlchemy, an extension that provides a Flask-friendly wrapper to the popular SQLAlchemy package, which is an Object Relational Mapper or **ORM**
To install Flask-SQLAlchemy, an extension

```
(venv) $ pip install flask-sqlalchemy
```

Flask-Migrate

Flask-Migrate is an extension that handles SQLAlchemy database migrations for Flask applications using Alembic. The database operations are provided as **command line** arguments for Flask-Script. To Install flask-migrate

```
(venv) $ pip install flask-migrate
```

Database Models

The data that will be stored in the database will be represented by a collection of **classes**, usually called database models.

Flask-SQLAlchemy Configuration

```
SQLALCHEMY_DATABASE_URI = 'sqlite:///' + os.path.join(basedir, 'app.db')
SQLALCHEMY_TRACK_MODIFICATIONS = False
```

and

```
db = SQLAlchemy(app)
migrate = Migrate(app, db)
```

CODE First Steps

```
1.  # Create Database Models in python
2.  flask db init
3.  flask db migrate -m "users table"
4.  flask db upgrade
```

SQLite

SQLite is a relational database management system contained in a C programming library. It stores data in file system.
The database file is able to opened in GUI application. Such as http://sqlitebrowser.org/

CODE first approach

Create database model in code first. Then generate the DDL SQL code using Migrate. Run update to create Database definitions.
After that, both database side and application side is able to run DML. Such as

```
insert into user values (3,'admin','admin@test.com','666666')
insert into post values(1, 'I love Python', 'this is a happy story', 'Python can do everything. However, it is very easy.',2)
insert into post values(2, 'I hate Python', 'this is a sad story', 'Python can do everything. However, it is very hard.',2)
```

Flask-Login

Installation

```
(venv) $ pip install flask-login
```

Password Hashing

```
>>> from werkzeug.security import generate_password_hash
>>> hash = generate_password_hash('foobar')
>>> hash
'pbkdf2:sha256:50000$vT9fkZM8$04dfa35c6476acf7e788a1b5b3c35e217c78dc04539d295f011f01f18cd2175f'
```

Using ORM for query database

```
# get the user from data base use code
        user = User.query.filter_by(username=form.username.data).first()
# get all post
posts = Post.query.all()
```

UserMixin

Added four required items for User, just not repeating some code already done by others.

- is_authenticated: a property that is True if the user has valid credentials or False Otherwise.

- is_active: a property that is True if the user's account is active or False otherwise.
- is_anonymous: a property that is False for regular users, and True for a special, anonymous user.
- get_id()